electronics

MDPI

Article

USB Artifact Analysis Using Windows Event Viewer, Registry and File System Logs

Ashar Neyaz †,‡ and Narasimha Shashidhar *,‡

Department of Computer Science, Sam Houston State University, Huntsville, TX 77341, USA; ashar.neyaz@shsu.edu

* Correspondence: karpoor@shsu.edu; Tel.: +1 (936)-294-1591

† Current address: 1900 Avenue I, Suite 214, Academic Bldg One, Huntsville, TX 77341, USA

‡ These authors contributed equally to this work.

Received: 30 September 2019; Accepted: 7 November 2019; Published: 9 November 2019

Abstract: A USB mass storage device yields a lot of artifacts when connected to a system. These artifacts are persistent in nature and are retained even after the system has been shut down and the information they contain may assist in carrying out forensic analysis on a suspect system. In this paper, we demonstrate how Windows Event Viewer can be used to find forensic artifacts in a suspect system for investigative purposes. We also discuss the potential that Windows registry holds to identify USB devices' information that have been connected to the system, to corroborate our findings from Windows Event Viewer. Finally, we use the Windows 10 file system to extract log details that contain the setup information of a USB device that was connected to the system the very first time, and obtain the necessary identifiers and time stamp details.

Keywords: USB storage; Windows Event Viewer; registry; operating system logs; digital evidence; forensic artifacts

1. Introduction

A Universal Serial Bus (USB) is a type of connection that allows multiple USB devices to be connected to a computer system [1]. USB devices come in all shapes, sizes and types such as scanner, printers, web-cameras, digital cameras, joysticks, USB hubs, WiFi dongles, keyboards, mice, flash storage drives, and external hard drives via USB interface and are being used everywhere around the globe. USB standard has altogether replaced serial and parallel ports and offers an unparalleled level of convenience to the user. USB has a number of advantages such as high speed data processing, hot swapping plug-and-play (PnP) support, and self-power supplying to peripheral devices [2]. Recently, there has been a considerable price drop in prices of USB devices and cost per storage today is much cheaper in comparison to what it was a decade ago [1].

As convenient as these devices are, they are also known for security risks. A USB device with a significant amount of storage capacity can download a large amount of sensitive information such as trade secrets and this information can later be exchanged for malevolent purposes. It can put the owner of the original information at great risk. In addition to providing the downloading of data, these USB devices are also responsible for bringing down systems by inserting malicious code, such as worms or trojans, and infecting them [1]. Regardless of what a system administrator or a security conscious computer user does, there is always a threat present in the use of USB peripherals [2].

When a USB device is attached to a system, operating system drivers start to collect information from the device and then use that information to create unique artifacts in the system itself, which is recorded. This collected information is persistently stored in the system, and in some cases is also consistent across different operating systems [1]. These unique artifacts can help in determining the

type of devices that was inserted into the system, give timeline information such as when was the device inserted, how many times the device was inserted, etc. This is valuable information for forensic investigators.

2. Background Information

The USB standard is responsible for defining a software protocol, firmware protocol, and a set of hardware used in communication between a host system and a USB device across a serial bus. USB is a master–slave protocol meaning that the host initializes all interactions. Today, almost every computing device offers USB support. This includes personal computers, servers, smartphone, tablets, routers, and embedded systems [3,4].

In order for any USB device to be used with a host, it goes through a mandatory setup procedure which consist of three steps. The first step is the bus setup, during which a set of standard electrical signals is relayed between the host and device's respective serial interface engines (SIE) [4]. It is this step that indicates to the host that a device is connected to the system. Then, these two parties go through a handshake protocol and negotiate parameters such as the communication speed of the device. The second step, which is the enumeration phase, is where the host queries the device to determine information such as the device's type (e.g., mass storage or human interface device), manufacturer and model, and the functionality it supports, among other parameters. Finally, in the last step, further interactions are passed from the host's client software through the standard system call interface (e.g., read(), write(), and ioctl()) to the device's high-level USB functions (e.g., providing an interface to internal storage, relaying video from a webcam) [4]. Figure 1 shows the enumeration phase of the USB data flow.

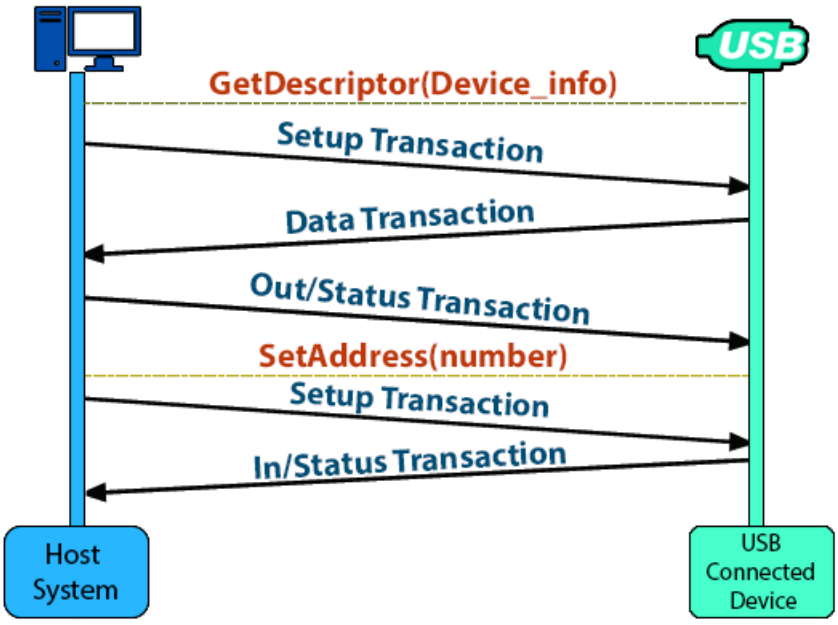

Figure 1. A USB data flow showing the enumeration phase highlighting the different transactions involved.

The enumeration process is a host-driven operation and it includes a three-layered protocol. At the top layer are control transfers: this step exchanges configuration information between the host system and the connected USB device [4]. The *GetDescriptor*(*device*) in the control transfer informs the host system of the USB device's manufacturer. In the middle layer, there is a setup transaction, which describes in detail what the subsequent transactions will be. Once the content of the setup transaction is known, the content of subsequent messages is well defined. The IRQ (interrupt requests) on a device must be compulsorily signaled at the end of each transaction to inform the software of the USB

device to queue a future transaction. Lastly, at the bottom layer, there are USB packets, which transmit the actual data. Each control transfer is formed by two or more transactions, and each transaction is composed of two or more packets [4].

3. Related Work

The use of USB devices is not going to decrease, but in fact is expected to rise in the coming years. Based on the trend of USB device adoption over the past few years, it is conceivable that the USB standard might supersede all other standards of connectivity such as that of ethernet or audio ports. Being the most popular standard in the market as of now, USB devices are constantly the focal point of attention by malicious users for malevolent purposes. These devices are being attacked, subverted, and scanned for vulnerabilities routinely. They are also being deployed as malicious carriers in an effort to bring down a system or an entire network.

Schumilo, Spenneber and Schwartke [5] implemented a USB fuzzer technique that can be used to secure USB drivers against vulnerable software flaws, which in turn improves security at the driver layer by improving the internal logic of device drivers. However, the USB fuzzer mechanism was unable to defend against the firmware level attack on the USB. Tian, Bates and Butler [6] observed the root cause of BadUSB. A BadUSB attack takes place via a USB storage device having malicious firmware. The attack has malicious functionalities that infect host machines by injecting malicious scripts and make the USB device run outside its apparent purpose. The authors also designed and implemented a GoodUSB mechanism to prevent the BadUSB attack by enforcing certain permission based on user experience. However, this was not implemented for the MS-Windows platform.

Wang, Stavrou [7] and Davis [8] observed some variations in the host-to-device identification protocols that leak useful information about the host operating system, respectively. However, this technique bowed down to an active fingerprint target that Bates et al. demonstrated [4]. On a different note, timing analysis of USB packets has also proven to be an effective measure in identifying the host operating system, as shown by Letaw, Pletcher and Butler [9].

Butler, McLaughlin and McDaniel presented a TPM attestation over USB interface for host verification [10]. However, the TPM module is still vulnerable to cuckoo attacks [11]. Angel et al. [12] designed a technique called Cinch, which serves as mechanism to thwart plug-and-play buses attacks. This technique needs a virtualized environment to function, however. Wang and Stavrou [7] proposed a USB-equipped smart phones mechanism to identify host operating systems. However, the approach of reading the contents of the USB Request Block (URB field) from packets was not effective against a well-versed adversary, who can manipulate data packets.

Pham et al. [2] analyzed the working mechanisms of U3 portable applications and USB platform free portable applications. Additionally, the authors also described the compilation process and techniques for creating U3 free portable applications that can be utilized by attackers to create their portable hack tools to execute from USB storage devices.

Carvey and Altheide [1] outlined the USB artifacts that are available in older Windows operating systems, namely Windows XP. Our work builds on this research and we take the lessons learned thus far in the literature and apply them to modern, current-generation operating systems, as explained in the next paragraph.

All the aforementioned work discusses either preventing the USB attacks from spreading, hardening the host device or designing a software tool. Our approach is novel in that we present a deeper exposition of the USB standard in the new Microsoft Windows 10 operating system and highlight the artifacts that can be found in an effort to conduct a sound forensic analysis. We make use of the Windows Event Viewer utility, native to the Windows platform, and the event ID object to classify a particular USB device as being inserted and/or removed. We further illustrate how to read this valuable treasure trove of information available in this latest Microsoft operating system to make a case for timeline analysis, which we hope will prove invaluable as part of a complete forensic examination.

4. Methodology and Analysis

This work is inspired by one of the research efforts by Carvey and Altheide [1]. They used an older Windows operating system (XP) to carry out their research work and outlined the USB artifacts and information by using the Windows registry editor, native to XP and other MS-Windows operating systems. To reduce the footprint of a digital forensic investigator on an evidence computing device, we divided the experiment into the following four parts:

Part 1: Setting up the Windows Event Viewer to obtain the USB drive information.
Part 2: Obtaining the USB information from Windows Registry by location the particular key associated with USB.
Part 3: Explaining the USB device characteristics using Device Manager property.
Part 4: Locating the setup information of USB device in the file system.

The above steps assist a digital forensic investigator in finding out the persistent USB artifacts on the system in question. The aim of performing these steps was to reduce the digital footprint of the investigator on the system while carrying out forensic analysis and also finding out the signature of USB devices.

- **Methodology Part 1: Windows Event Viewer Setup based on event IDs**

The **Windows Event Viewer** has two log categories, namely **Windows Logs** and **Applications and Services Logs**. These two categories have different sub-categories inside them. The Windows Logs has *Application, Security, Setup, System and Forwarded Events*, whereas Applications and Services Logs has *Hardware Events, Internet Explorer, Key Management Service, Microsoft, OpenSSH, and Windows Powershell*. The Windows event viewer records all the events taking place in a system and stores them in a particular category type with different event level description. The following is a list of the event level descriptions [13]:

- Information: Displays the non-critical information.
- Warning: Provides forewarning of potential problems and not actual errors. A warning indicates that a component or application is not in an ideal state and that some further actions could yield a critical error.
- Verbose: Displays progress or success messages
- Critical: Displays the information that requires immediate attention of the system administrator. They can also be used to indicate that an application or system has failed or stopped responding.
- Error: Displays events that indicate problems, but immediate action is not required.

For USB device connection and disconnection log, the **Applications and Services Logs** is used. All the event related to USB are recorded here with particular event ID for distinction. The following is a list of the event IDs associated with USB device connection and disconnection:

- **–Connection**
- 2003
- 2004
- 2006
- 2010
- 2101
- 2105
- 2106
- **–Disconnection**
- 2100
- 2102

Since there are multiple event IDs associated for USB connection and disconnection, for convenience when conducting the experiment, the IDs 2003 and 2102 were chosen to find information about a USB device along with its timestamp. However, the USB logging in Windows Event Viewer is not enabled by default, thus, to enable it, the Event Viewer was opened and the following path was traversed, Application and Service Logs -> Microsoft -> Windows -> DriverFrameworks-UserMode -> Operational. Then, we right-clicked on Operational, checked Enable Logging in the Properties and then clicked OK [14]. Figure 2 displays the enable logging step. The screenshots in Figures 2–11 outline the USB thumb drive analysis.

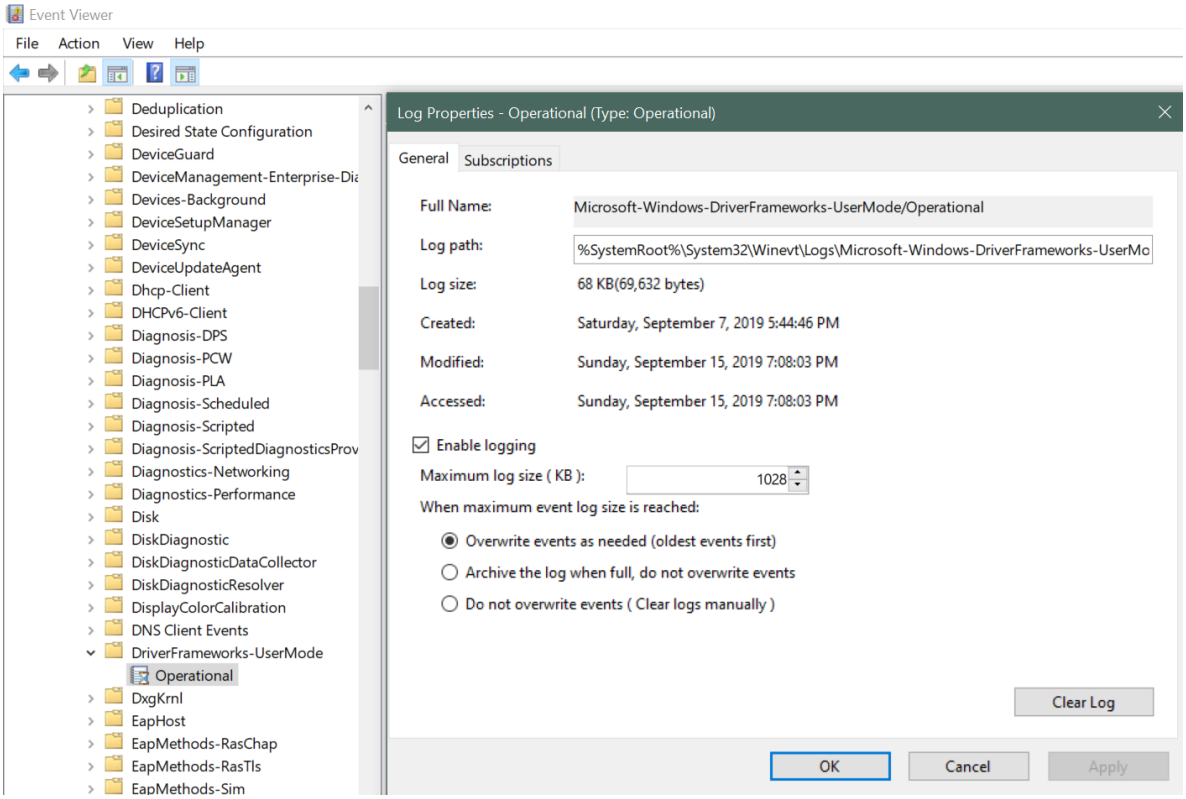

Figure 2. Enable USB logging in Event Viewer.

Figures 3–8 show information obtained for the USB flash drive when inserted and disconnected from the Windows 10 operating system. The textual information from the screenshots are summarized below, describing the characteristics of a USB device:

a. When the vendor is generic:

1. **Event ID**: 2003 (for device connection)
2. **USB type**: USB flash drive indicated by *USBSTOR#DISK*
3. **Vendor name**: Generic (as shown) indicated by *VEN_GENERIC*, upon further investigation on devicehunt.com, vendor was found to be Alcor Micro Corp.
4. **Product type**: Flash Disk indicated by *PROD_FLASH_DISK*
5. **Serial number**: *5991D10D*
6. **Connection timestamp**: *15 September 2019 at 6:47:13 PM*
7. **Event ID**: 2102 (for device disconnection)
8. **Disconnection timestamp**: *15 September 2019 at 7:03:02 PM*
9. **Disconnection status code**: 0x0

b. When the vendor is specific: (exact vendor and product names were mentioned in the event viewer)

1. **Event ID**: 2003 (for device connection)
2. **USB type**: USB flash drive indicated by *USBSTOR#DISK*
3. **Vendor name**: Kingston indicated by *VEN_KINGSTON*
4. **Product type**: Data Traveler 3.0 indicated by *PROD_DATATRAVELER_3.0*
5. **Serial number**: *5404A6F4E0A6BF81492A00E2*
6. **Connection timestamp**: *15 September 2019 at 7:50:41 PM*
7. **Event ID**: 2102 (for device disconnection)
8. **Disconnection timestamp**: *15 September 2019 at 7:51:06 PM*
9. **Disconnection status code**: 0x0

For the purpose of convenience, we also made a customized view of USB tracking mechanism using the Microsoft Event Viewer track down devices. Figures 9–12 outline the steps involved in creating the custom view. Only two different USB flash drives were used for conducting the experiment, as it was known that the characteristics associated to USB flash drives are similar if not the same and the experiment proved it. Having multiple USB flash drives would not have yielded any new information overall. Table 1 summarizes the description of figures of Part 1 of the methodology.

Table 1. The description of figures for Methodology Part 1.

Figures	Description
Figure 3	This figure displays overall view of the Operational log with many event IDs along with ID 2003, which corresponds to USB device connection to the computer system.
Figure 4	This figure describes the information related to the Generic USB drive corresponding to event ID 2003 in the Windows Event Viewer. The figure clearly shows the USB device signature along with the log-in time, event ID number, log name, log level, category and the computer name.
Figure 5	This figure describes the information related to the Kingston USB drive corresponding to event ID 2003 in the Windows Event Viewer. The figure displays the USB device signature along with the log-in time, event ID number, log name, log level, category and the computer name.
Figure 6	This figure displays overall view of the Operational log with many event IDs along with ID 2102 which corresponds to USB device disconnection from the computer system.
Figure 7	This figure describes the information related to the Generic USB drive corresponding to event ID 2102 in the Windows Event Viewer. Similar to Figure 3, it shows the USB device signature along with the log-out time, event ID number, log name, log level, category and the computer name.
Figure 8	This figure describes the information related to the Kingston USB drive corresponding to event ID 2102 in the Windows Event Viewer. The figure displays the USB device signature along with the log-out time, event ID number, log name, log level, category and the computer name.
Figure 9	The figure describes steps involved in the Create Custom View window of Windows Event Viewer.
Figure 10	This figure displays the naming of the custom view filter. In our work, we named it as USB Flash Drive Connect-Disconnect Tracker.
Figure 11	The figure displays the custom view sorted by event ID.
Figure 12	The figure displays the custom view grouped by event ID for better readability.

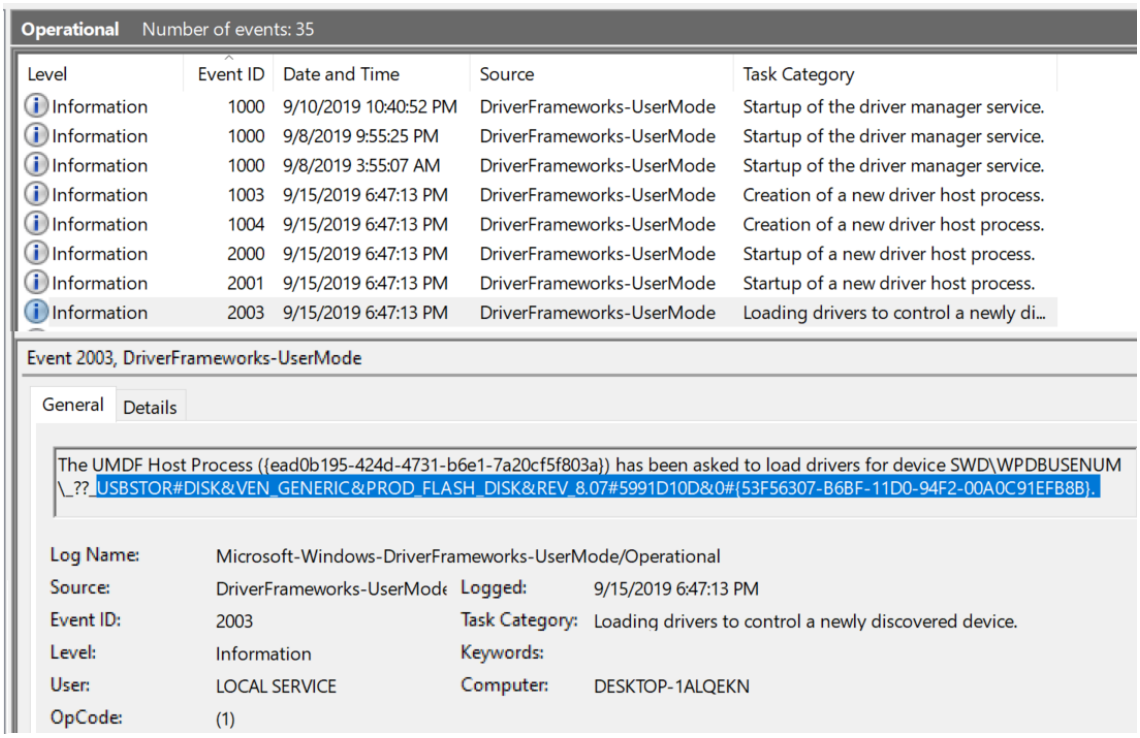

Figure 3. Event ID 2003 in Windows Event Viewer.

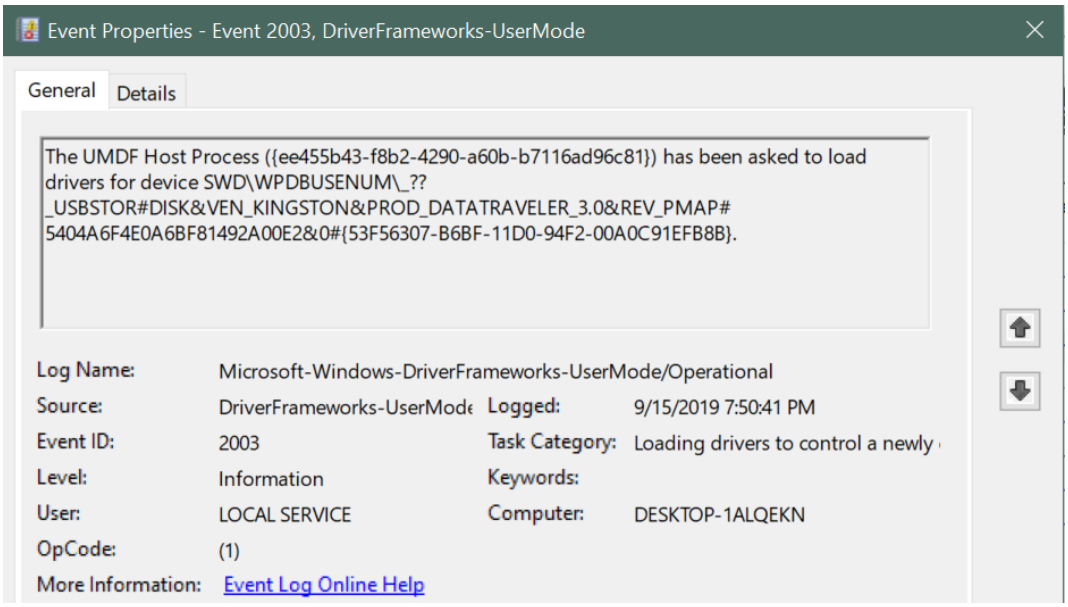

Figure 4. Event ID 2003 for Kingston USB drive.

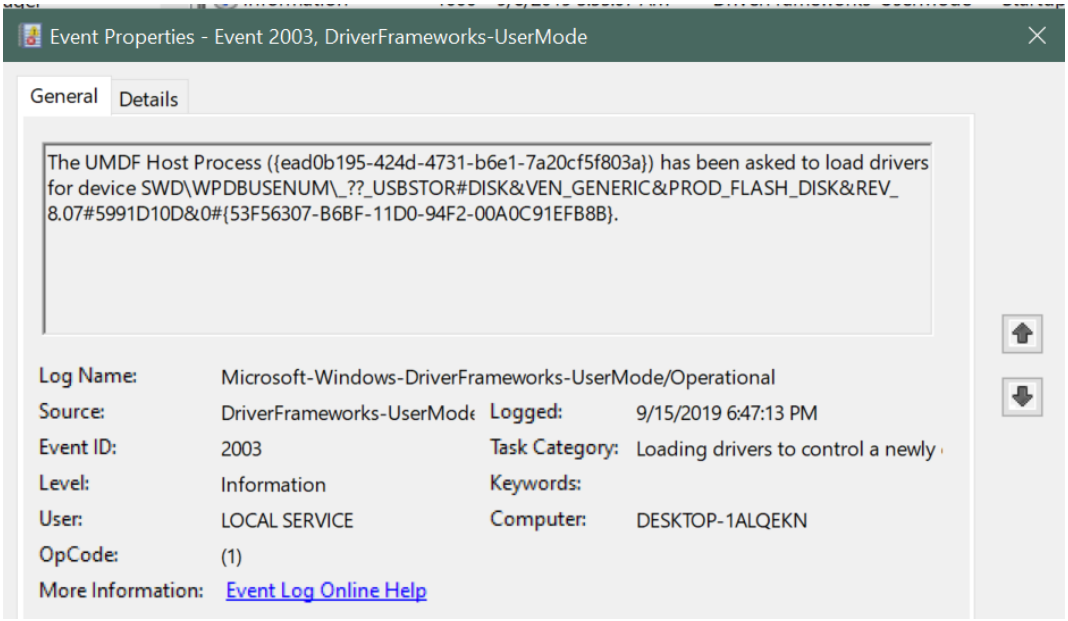

Figure 5. Event ID 2003 for generic USB drive.

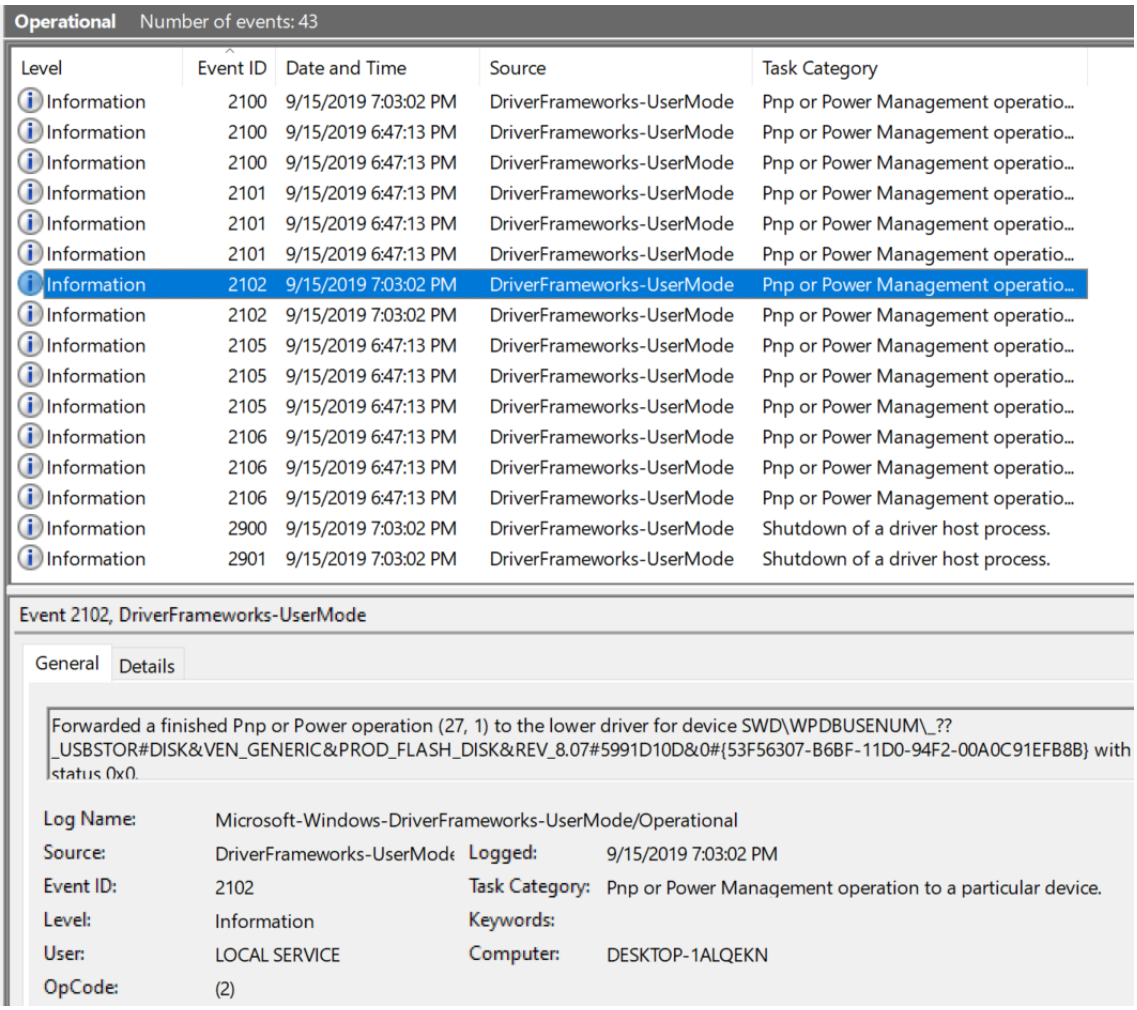

Figure 6. Event ID 2102 in Windows Event Viewer.

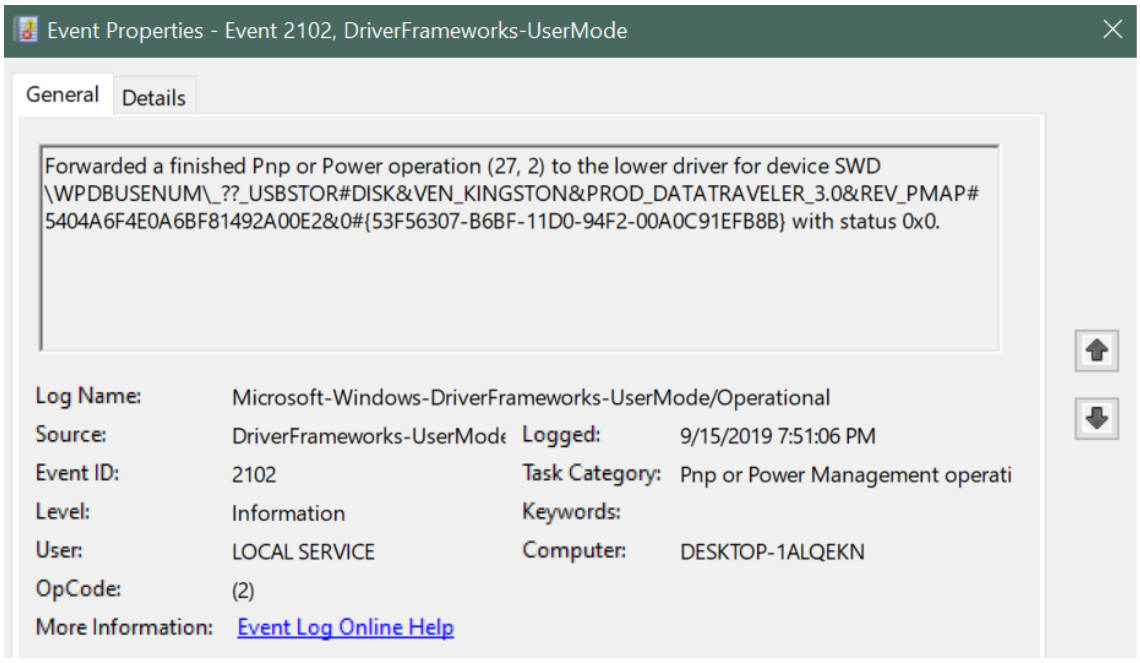

Figure 7. Event ID 2102 for Kingston USB drive.

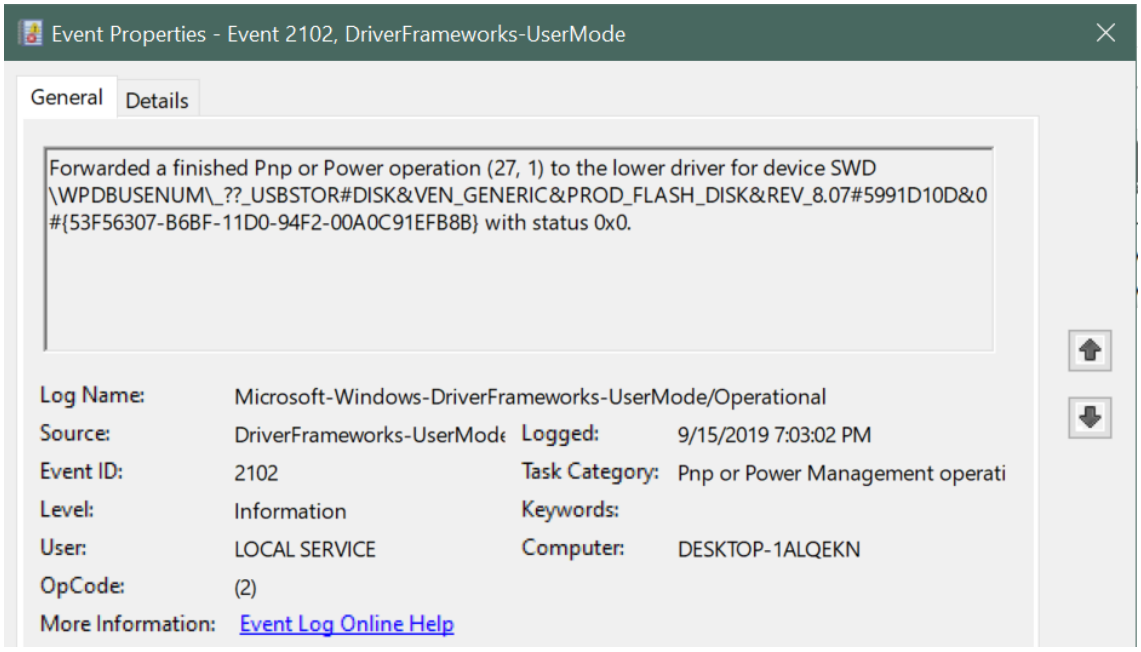

Figure 8. Event ID 2102 for generic USB drive.

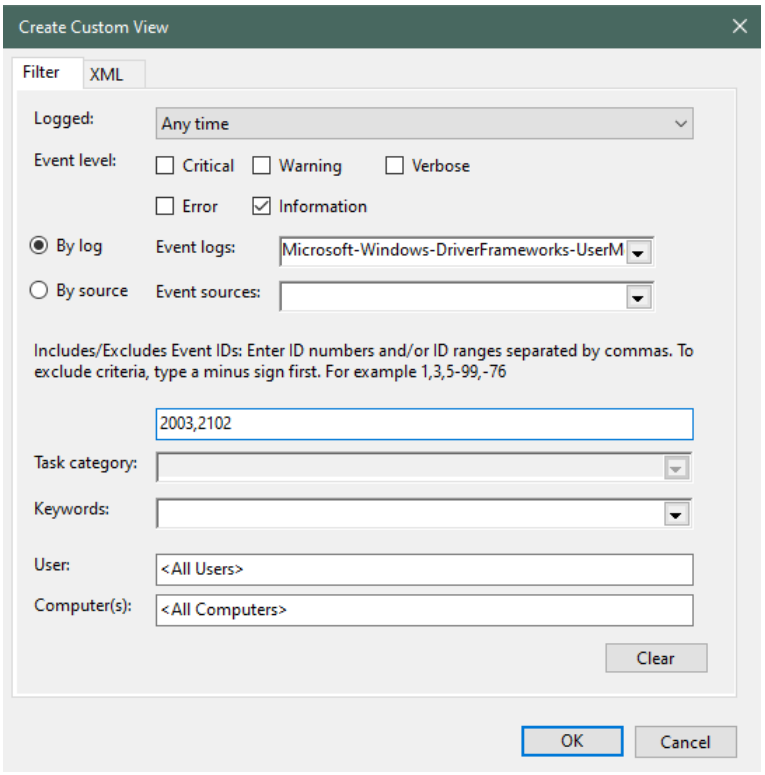

Figure 9. Customizing view of Windows Event Viewer.

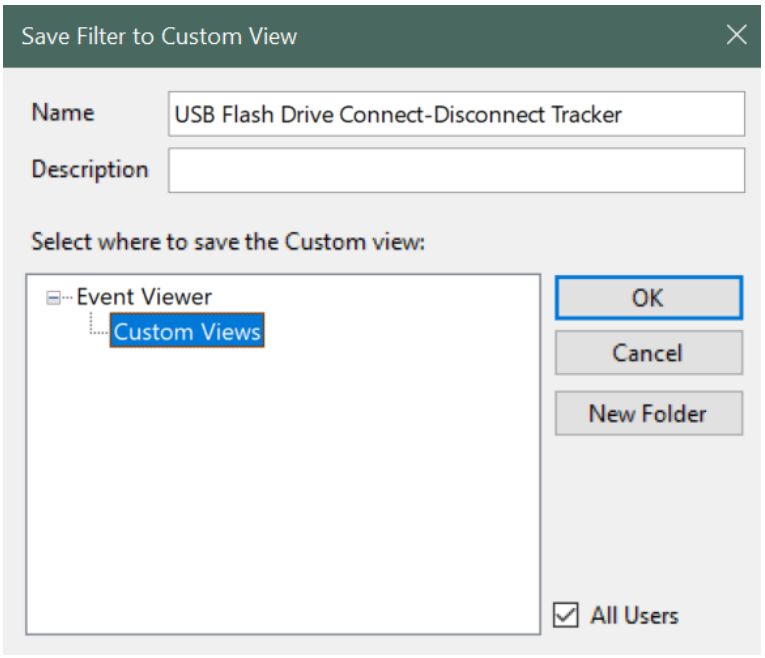

Figure 10. Naming the custom filter of Windows Event Viewer.

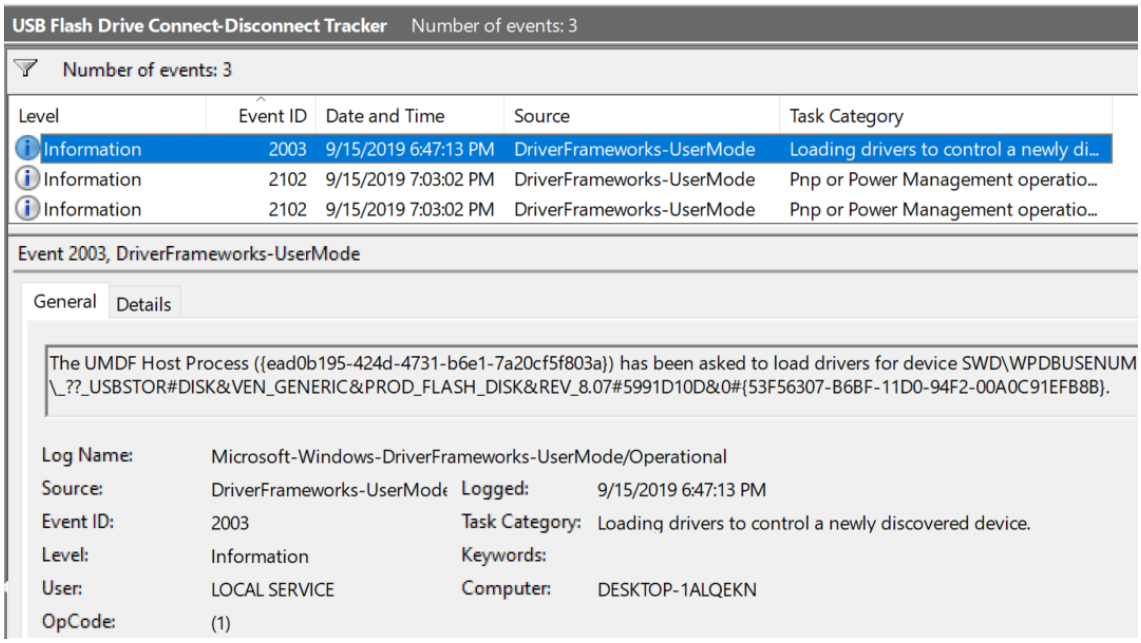

Figure 11. Custom view for event IDs.

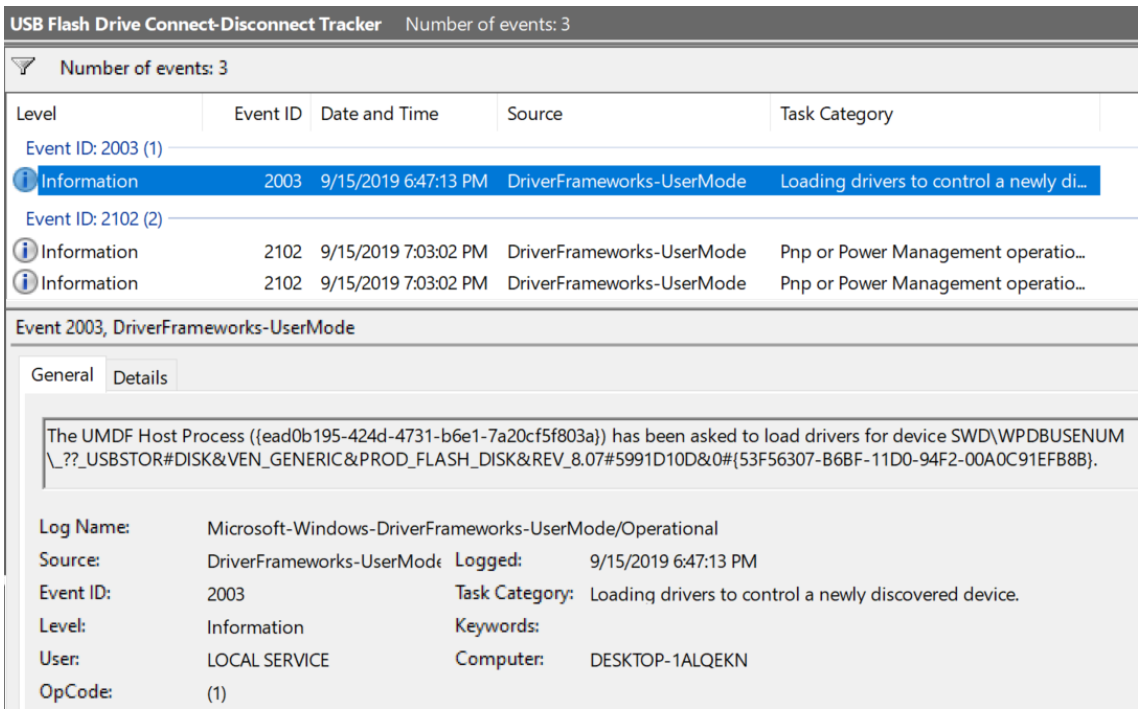

Figure 12. Group by view for event IDs.

- **Methodology Part 2: Device manager and USB device properties**

The **Windows Device Manager** is a member of the Microsoft Management Console that serves as central repository and provides an organized view of all the recognized hardware installed in a Windows computer. The device manager is used manage the installed hardware peripherals such as USB devices, keyboards, sounds, storage drives, graphics card, etc. [15].

As far as Windows Device Manager information is concerned, in regards to a USB device, the following screenshot (Figure 13) explains the USB flash drive information that was obtained from ***Device Instance Path*** in USB Mass Storage Device Properties window [1].

To see the information or identifiers related to the connected USB, the following steps were taken:

1. The Device Manager was opened by going to the Windows Control Panel while a USB device was still connected to the system.
2. The appropriate device was selected, in this case USB Mass Storage Device under the Universal Serial Bus Controller.
3. Then, the USB Mass Storage Device was right-clicked to select the Properties.
4. Lastly, the Device Instance Path was selected from the Details tab of the USB device.

[**Note**: Hardware IDs or Compatible IDs can also be selected from the drop-down list for similar USB information.]

Windows creates Device Instance Path in the following syntactical format: **USB\VID_v(4)&PID_d(4)&Serial_No**

For Generic USB Flash Drive: **USB\VID_058F&PID_6387&5991D10D**

For Kingston USB Flash Drive: **USB\VID_0951&PID_1666&5404A6F4E0A6BF81492A00E2**

The list below shows the breakage of information that was obtained from Device Instance Path, as shown in Figure 13. [**Note**: Figure 13 only shows the Generic USB Flash Drive information. Similar syntactical information can be obtained for the Kingston Flash Drive.]

-Generic USB Flash information:

1. VID (Vendor ID): 058F
2. PID (Product ID): 6387
3. Serial Number: 5991D10D

-Kingston USB Flash information:

1. VID (Vendor ID): 0951
2. PID (Product ID): 1666
3. Serial Number: 5404A6F4E0A6BF81492A00E2

- **Methodology Part 3: Windows Registry Editor information**

The **Windows registry** is a hierarchical database that contains critical installed information about hardware and software component that is essential for smooth running of application and services on Windows operating system. The database is structured in a tree style format where each node in the tree is called a hive. A hive contains many keys and each key can contain both subkeys and data entries called values [16].

The following list out the hives in Windows Registry Editor:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
- HKEY_USERS
- HKEY_CURRENT_CONFIG

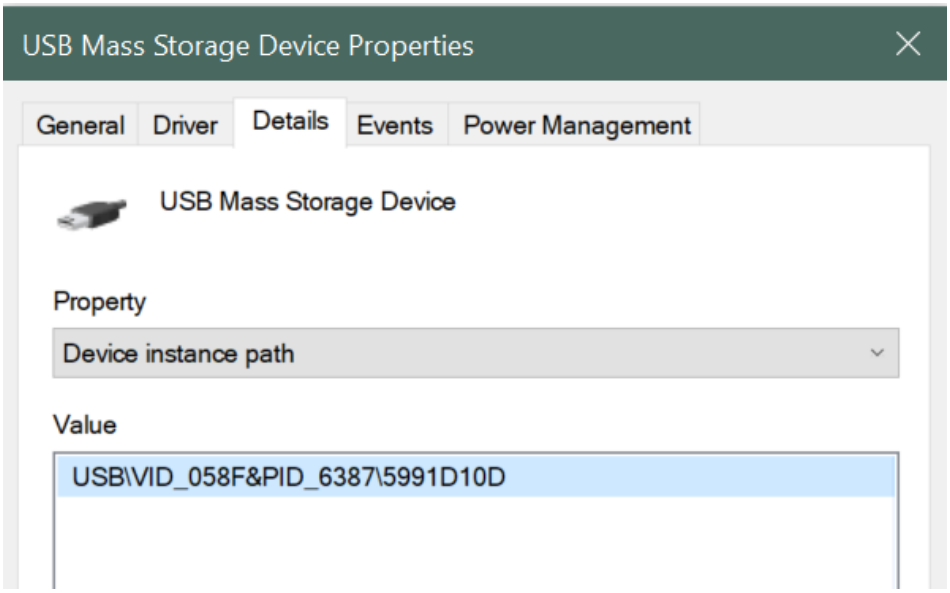

Figure 13. Device Instance Path of the USB Mass Storage Device.

The point of interest for successful USB device setup information is located in the following registry keys:

Firstly, ***HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB***: A series of subkeys are present under this registry key based on device ID as shown in USB Mass Storage Device Properties. Each of device ID subkeys has one or more subkeys present which describe the device's parameters. Figure 14 shows the USB key that contains information about the Vendor ID(VID), Product ID(PID), and Serial Number. The VID and PID are bundled as one subkey, which in turn has a Serial Number key inside it.

The Serial Number key has useful values in it such as:

1. *DeviceDesc*: Information about the devise description. The value contains **USB Mass Storage Device**, indicating that it was a USB flash drive.
2. *Driver*: The driver information that was used in installing the USB device.
3. *HardwareID*: This contains the hardware signature values such as vendor ID and product ID.
4. *LocationInformation*: This value displays the port and hub number to which the USB device was attached to on the system. In the figure, the USB device was attached to Port 4, Hub3.
5. *Mfg*: This displays the name of Setup Information File or the INF file, which in this case was usbstor.inf, and status of the compatibility of USB device, which was Compatible USB storage device.
6. *Service*: Display the service name associated with this registry case. In this case, the service name was USBSTOR.

Secondly, ***HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USBSTOR***: The information present under the subkeys of this registry key was less than the USB subkey as USBSTOR key is only specific for USB mass storage device, whereas USB key is meant for USB devices connected to system in general [1].

Figure 15 shows the USBSTOR key. In this case, USBSTOR key had a subkey Disk&Ven_Generic, which in turn had another subkey inside it labeled as the serial number of the USB Flash Drive. This subkey had similar information as was present in the USB key. The following is a list of the information present in the USBSTOR key:

1. *DeviceDesc*: Information about the devise description. The value contains **disk drive**, indicating that it was a USB flash drive.
2. *FriendlyName*: The device name for the device. In this case, it was Generic Flash Disk USB Drive.

3. *Mfg*: This displays the name of Setup Information File or the INF file, which, in this case was disk.inf, and the type of USB device, which was Standard disk drives.
4. *Service*: This displays the service name associated with this registry case. In this case, the service name was disk.

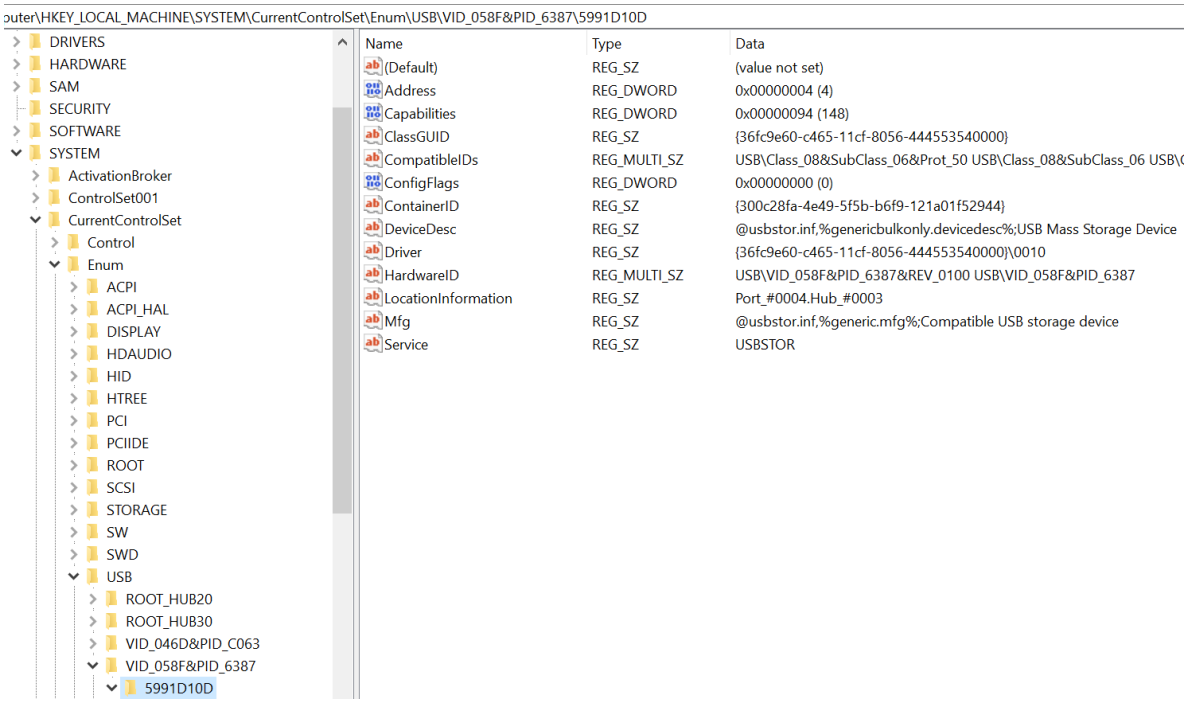

Figure 14. USB Registry key.

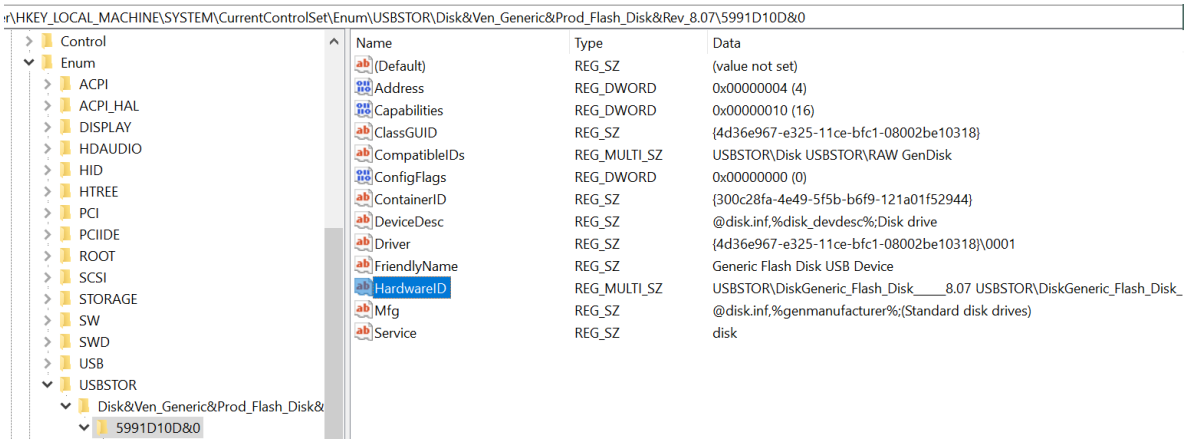

Figure 15. USBSTOR registry key location.

Moreover, by going through the list of values shown in **HKEY_LOCAL_MACHINE\SYSTEM\MountedDevices**, the associated drive letter, GUID, name, vendor ID, product ID and serial number of the USB mass storage device were found.

Figures 16–18 show the list of mounted devices, their associated drive letters, and the values of the Generic USB mass storage device that was connected to the system.

1. **GUID**: 8bc1b2bf-4902-11e9-9f78-b8763fe48ac0
2. **USB Type**: USB flash drive indicated by USBSTOR#DISK
3. **Vendor name**: Generic
4. **Product type**: Flash Disk
5. **Serial number**: 5991D10D
6. **Drive Letter**: F:

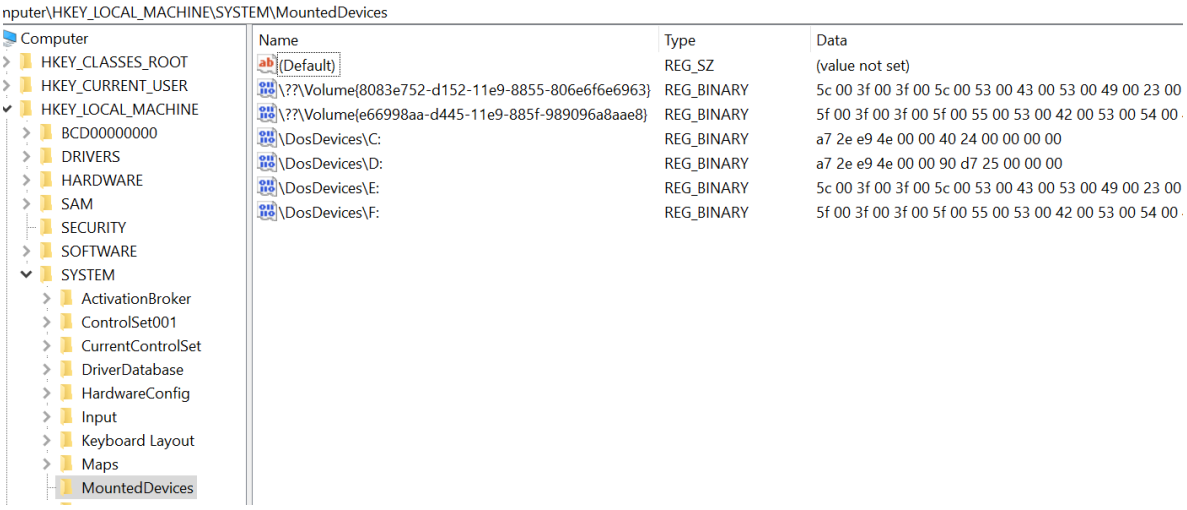

Figure 16. USB device information.

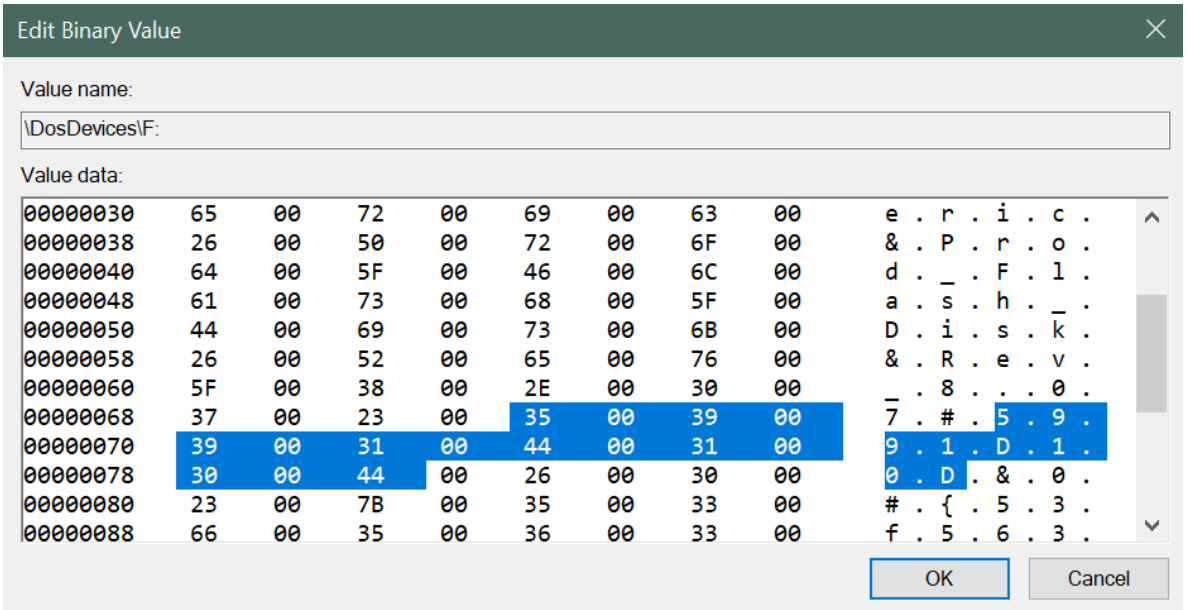

Figure 17. USB device information.

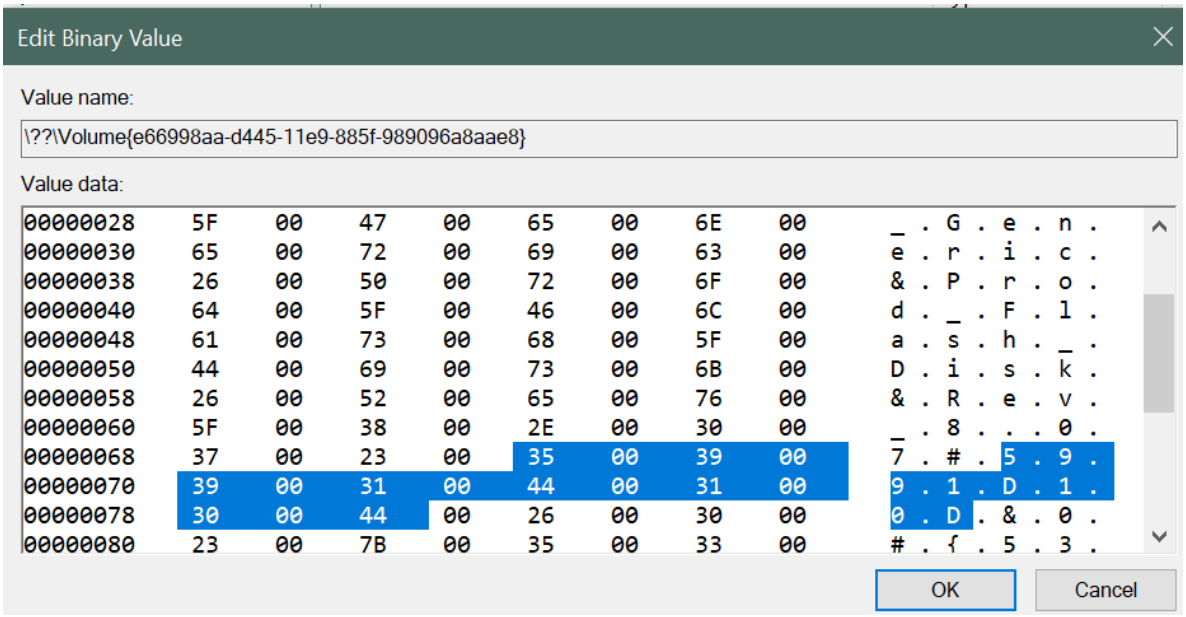

Figure 18. USB device GUID.

However, the drive letters are not unique or tied to a particular USB mass storage device. Windows can assign the same drive letter F: to some other USB mass storage device when it is connected in the future and will assign some other drive letter to the previously connected USB mass storage device.

- **Methodology Part 4: Location of USB setup information from the file system**

Microsoft Windows have a log file dedicated to logging all the information related to a USB device when it was connected to the system for the very first time. To locate the log file, the following path was traversed: **C:>\Windows\inf\setupapi.dev.log**. This log file contains detailed information regarding the USB device setup and also the name and version of the operating system to which the USB device was connected to.

Figures 19 and 20 show excerpts from the setupapi.dev.log file for the ***Generic USB mass storage device*** and ***Kingston USB mass storage device***. Windows 10 does not log vendor ID and device or product ID, as it did previously in the case of Windows 7. With Windows 10 operating system, the log only saves the vendor name, product name and serial number of the device associated.

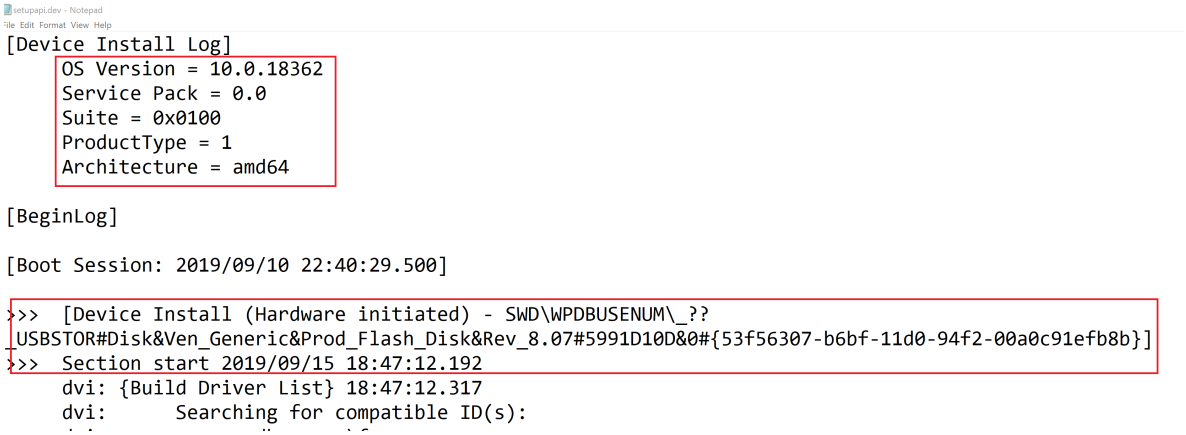

Figure 19. Generic USB setup information in setupapi.dev log file.

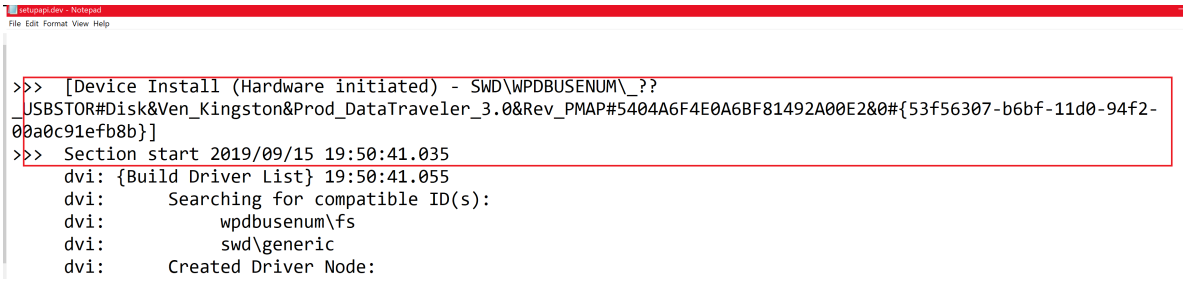

Figure 20. Kingston USB setup information in setupapi.dev log file.

5. Conclusions and Future Work

Analyzing Windows event viewer, registry and file system log help in identifying a USB device's identifiers such as product ID, vendor name, serial numbers, and operating system version. The information is not easy to comprehend at once. In this paper, we thoroughly analyze the Windows services, i.e., the event viewer, registry, and log, and make a correlation among them to assist in digital forensic investigation by locating the useful USB artifact information.

Furthermore, we also explain the integrity of the information presented by Windows operation system. The artifacts obtained from Windows Event Viewer, Windows Registry, Device Manager and setupapi.dev log file show no change in the USB device's signature information, implying that no malicious activities had taken place on the system in order to obscure the digital forensic footprints. If all the information did not match, there would have been an indication that an anomaly had happened and the findings would not have been correct or accepted in the court of law.

This work was immensely inspired by one of the works of Harley Carvey [1] that helped us in reaching this point. In addition, this work opens an interesting avenue for further analyzing different types of USB devices in different operating systems. The future work will involve working on newer Linux and Apple operating systems to forensically analyze USB artifacts for forensic examination.

Author Contributions: Conceptualization, A.N.; methodology, A.N.; validation, N.S.r; supervision, N.S.; writing-original draft preparation: A.N. and N.S.; formal analysis, N.S.

Funding: This research work received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Carvey, H.; Altheide, C. Tracking USB storage: Analysis of windows artifacts generated by USB storage devices. *Digit. Investig.* **2005**, *2*, 94–100.
2. Pham, D.V.; Syed, A.; Mohammad, A.; Halgamuge, M.N. Threat analysis of portable hack tools from USB storage devices and protection solutions. In Proceedings of the 2010 International Conference on Information and Emerging Technologies, Karachi, Pakistan, 14–16 June 2010; pp. 1–5.
3. Universal Serial Bus Specification. Available online: http:sdphca.ucsd.edulab_equip_manualsusb_20.pdf (accessed on 7 August 2019).
4. Bates, A.; Leonard, R.; Pruse, H.; Lowd, D.; Butler, K.R. Leveraging USB to Establish Host Identity Using Commodity Devices. In Proceedings of the NDSS, San Diego, CA, USA, 23–26 February 2014.
5. Schumilo, S.; Spenneberg, R.; Schwartke, H. Don't trust your USB! How to find bugs in USB device drivers. In Proceedings of the Blackhat Europe, Amsterdam, The Netherlands, 14–17 October 2014.
6. Tian, D.J.; Bates, A.; Butler, K. Defending against malicious USB firmware with GoodUSB. In Proceedings of the 31st Annual Computer Security Applications Conference, Los Angeles, CA, USA, 7–11 December 2015; pp. 261–270.
7. Wang, Z.; Stavrou, A. Exploiting smart-phone usb connectivity for fun and profit. In Proceedings of the 26th Annual Computer Security Applications Conference, Austin, TX, USA, 6–10 December 2010; pp. 357–366.
8. Davis, A. Revealing embedded fingerprints: Deriving intelligence from USB stack interactions. In Proceedings of the Blackhat, Las Vegas, NV, USA, 27 July–1 August 2013.

9. Letaw, L.; Pletcher, J.; Butler, K. Host identification via usb fingerprinting. In Proceedings of the 2011 Sixth IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering, Oakland, CA, USA, 26 May 2011; pp. 1–9.
10. Butler, K.R.; McLaughlin, S.E.; McDaniel, P.D. Kells: A protection framework for portable data. In Proceedings of the 26th Annual Computer Security Applications Conference, Austin, TX, USA, 6–10 December 2010; pp. 231–240.
11. Parno, B. Bootstrapping Trust in a "Trusted" Platform. In Proceedings of the HotSec, Berkeley, CA, USA, 29 July 2008.
12. Angel, S.; Wahby, R.S.; Howald, M.; Leners, J.B.; Spilo, M.; Sun, Z.; Blumberg, A.J.; Walfish, M. Defending against malicious peripherals with Cinch. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 397–414.
13. Microsoft Document. Trace and Event Log Severity Levels. Available online: https://docs.microsoft.com/en-us/previous-versions/office/developer/sharepoint-2010/ff604025(v%3Doffice.14) (accessed on 7 August 2019).
14. Shultz, G. How to Track down USB Flash Drive Usage with Windows 10's Event Viewer. Available online: https://www.techrepublic.com/article/how-to-track-down-usb-flash-drive-usage-in-windows-10s-event-viewer (accessed on 7 August 2019).
15. Fisher, T. What Is Device Manager. Available online: https://www.lifewire.com/device-manager-2625860 (accessed on 7 August 2019).
16. Microsoft Document. Structure of the Registry. Available online: https://docs.microsoft.com/en-us/windows/win32/sysinfo/structure-of-the-registry (accessed on 7 August 2019).

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).